JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11)Publication number: 05342012
(43)Date of publication of application: 24.12.1993

(51)Int.Cl.
G06F 9/45
G06F 15/347

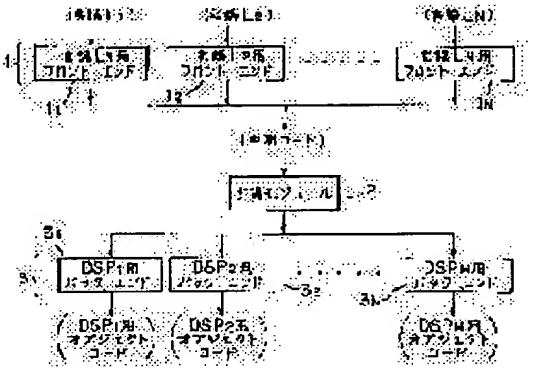
(21)Application number: 04176273
(22)Date of filing: 10.06.1992

(71)Applicant: SONY CORP
(72)Inventor: IWATA EIJI

(54) COMPILING METHOD AND COMPILER

(57)Abstract:

PURPOSE: To compile a program without depending on digital signal processors(DSP) and languages.
CONSTITUTION: At a front end part in for language Ln, corresponding to the language Ln [(n)=1, 2...N] preprocessing is performed to the source code of the program written in the language LN, and an intermediate code common' for languages L1-Ln is outputted. At a common module part 2, processing is performed to the intermediate code without depending on the languages L1-LN and DSP1-DSPM. At a back end part 3M [(m)=1, 2...M] for DSPM, post- processing is performed to the output of the common module part 2, and an object code is outputted corresponding to the DSPM.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

MENU SEARCH INDEX

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平5－342012

(43)公開日　平成５年(1993)12月24日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/45				
15/347		G　8320－5L		
		9292－5B	Ｇ０６Ｆ　9/44　322　Ａ	

審査請求　未請求　請求項の数５（全 11 頁）

(21)出願番号　特願平4－176273

(22)出願日　平成４年(1992)６月10日

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号

(72)発明者　岩田　英次
東京都品川区北品川６丁目７番35号　ソニー株式会社内

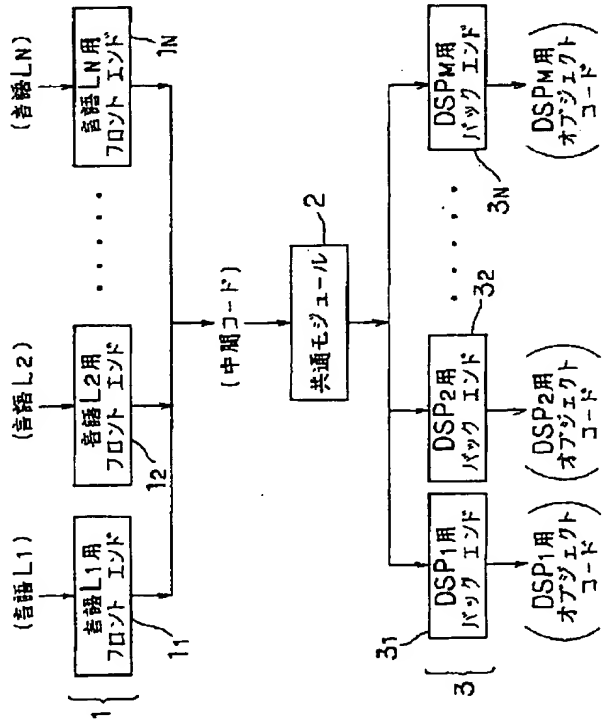
(74)代理人　弁理士　稲本　義雄　（外１名）

(54)【発明の名称】　コンパイル方法およびコンパイラ

(57)【要約】

【目的】　ディジタルシグナルプロセッサおよび言語に依存せずに、プログラムをコンパイルする。

【構成】　言語L_n用フロントエンド部1_nにおいて、言語L_n（n＝１，２，・・・，Ｎ）で記述されたプログラムのソースコードに対して、言語L_nに対応した前処理が行われ、言語L_1乃至L_Nに共通の中間コードが出力される。共通モジュール部２において、その中間コードに対して、言語L_1乃至L_NおよびDSP_1乃至DSP_Mに依存しない処理が行われ、ディジタルシグナルプロセッサDSP_m用バックエンド部3_m（m＝１，２，・・・，Ｍ）において、共通モジュール部２の出力に対して後処理が施されて、ディジタルシグナルプロセッサDSP_mに対応したオブジェクトコードが出力される。



【特許請求の範囲】
【請求項１】　ディジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換し、
前記中間コードを前記ディジタルシグナルプロセッサに対応したオブジェクトコードに変換するコンパイル方法において、
前記中間コードは、前記ディジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のディジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通であることを特徴とするコンパイル方法。
【請求項２】　ディジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換する前処理手段と、
前記前処理手段より出力される中間コードを解析して最適化する共通処理手段と、
前記共通処理手段からの出力を前記ディジタルシグナルプロセッサに対応したオブジェクトコードに変換する後処理手段とを備え、
前記前処理手段は、前記ディジタルシグナルプロセッサのプログラムを記述するための複数の言語に共通の中間コードを出力し、
前記後処理手段は、前記中間コードを複数のディジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに変換することを特徴とするコンパイラ。
【請求項３】　前記共通処理手段は、前記中間コードに対して、前記複数のディジタルシグナルプロセッサのどれにも依存しない最適化処理を施すことを特徴とする請求項２に記載のコンパイラ。
【請求項４】　前記後処理手段は、前記中間コードを並列に実行可能な単位に分割し、前記ディジタルシグナルプロセッサに割り当てることを特徴とする請求項２または３に記載のコンパイラ。
【請求項５】　前記後処理手段は、レジスタ割付を行うことを特徴とする請求項２，３、または４に記載のコンパイラ。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明は、例えば音声処理や画像処理などのディジタル信号処理に用いられるＤＳＰ（ディジタルシグナルプロセッサ）のプログラムをコンパイルする場合に用いて好適なコンパイル方法並びにコンパイラに関する。
【０００２】
【従来の技術】近年の音声処理や画像処理などにおいては、多くの場合、信号（音声信号や画像信号）をサンプリングし、即ち信号をディジタル化し、例えばＤＦＴ（離散フーリエ変換），ＦＦＴ（高速フーリエ変換）、またはＤＣＴ（離散コサイン変換）などにより、時間軸上の信号を周波数軸上の信号（スペクトル）に変換して、パラメータの抽出や圧縮符号化処理が行われる。
【０００３】このような、ディジタル信号の時間軸と周波数軸との相互変換（ＤＦＴ（逆ＤＦＴ），ＦＦＴ（逆ＦＦＴ）、およびＤＣＴ（逆ＤＣＴ）など）処理に代表されるディジタル信号処理においては、積和演算の回数が非常に多く、この処理を、例えば汎用的なＣＰＵなどで実現した場合には、実時間処理を行うことが困難であった。
【０００４】そこで、いわゆるパイプライン処理や並列処理に適したアーキテクチャを有し、積和演算を、例えば数１０乃至数１００ｎ秒で行うことのできる、いわば積和演算処理を得意とするＤＳＰ（ディジタルシグナルプロセッサ）が開発され、近年、このＤＳＰを用いた多くのディジタル信号処理装置（例えば画像処理装置や音声処理装置など）が実現されている。
【０００５】
【発明が解決しようとする課題】ところで、このＤＳＰに、ディジタル信号処理を実行させるには、まず、そのＤＳＰ用の言語でディジタル信号処理プログラムを記述し、そのソースコードをコンパイラでコンパイルしてオブジェクトコードに変換する必要がある。
【０００６】しかしながら、ディジタル信号処理プログラムを記述するための言語は、ＤＳＰにより異なる場合が多く、また同じＤＳＰであってもプログラムを記述するための言語が複数種類提供されている場合もあり、従って、例えばＭ個のＤＳＰのプログラムが、Ｎ種類の言語で記述することができるときには、各ＤＳＰおよび各言語に対応したＭ×Ｎ個のコンパイラを用意しなければならず、不便であった。
【０００７】なお、言語がＤＳＰにより異なる場合とは、言語の種類がＤＳＰにより異なる場合の他、同じ言語であってもその系統がＤＳＰにより異なる場合（例えば、コンピュータ言語でいえば、同じＣ言語でも、コンパイラの提供メーカにより異なる場合）を含む。
【０００８】さらに、コンパイラは、プログラムをコンパイルする過程を分割した、複数の論理的な操作単位（フェーズ）よりなっているが、上述した各ＤＳＰおよび各言語に対応した各コンパイラには、ＤＳＰおよび言語に依存しない、互いに共用することができるフェーズが、少なからず含まれており、各コンパイラにこのフェーズを設けることは無駄であった。
【０００９】本発明は、このような状況に鑑みてなされたものであり、ＤＳＰおよび言語に依存せずに、プログラムをコンパイルすることができるようにするものである。
【００１０】
【課題を解決するための手段】請求項１に記載のコンパイル方法は、ディジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換し、中間コードをディジタルシグナルプロセッサに対応したオブジェクト

コードに変換するコンパイル方法において、中間コードは、ディジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のディジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通であることを特徴とする。

【００１１】請求項２に記載のコンパイラは、ディジタルシグナルプロセッサＤＳＰ$_1$乃至ＤＳＰ$_M$のプログラムのソースコードを中間コードに変換する前処理手段としてのフロントエンド部１（言語Ｌ$_1$用フロントエンド部１$_1$乃至言語Ｌ$_N$用フロントエンド部１$_N$）と、フロントエンド部１より出力される中間コードを解析して最適化する共通処理手段としての共通モジュール部２と、共通モジュール部２からの出力をディジタルシグナルプロセッサＤＳＰ$_1$乃至ＤＳＰ$_M$に対応したオブジェクトコードに変換する後処理手段としてのバックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）とを備え、フロントエンド部１（言語Ｌ$_1$用フロントエンド部１$_1$乃至言語Ｌ$_N$用フロントエンド部１$_N$）は、ディジタルシグナルプロセッサＤＳＰ$_1$乃至ＤＳＰ$_M$のプログラムを記述するための複数の言語Ｌ$_1$乃至Ｌ$_N$に共通の中間コードを出力し、バックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）は、中間コードを複数のディジタルシグナルプロセッサＤＳＰ$_1$乃至ＤＳＰ$_M$にそれぞれ対応したオブジェクトコードに変換することを特徴とする。

【００１２】請求項３に記載のコンパイラは、共通モジュール部２に、中間コードに対して、ＤＳＰ$_1$乃至ＤＳＰ$_M$のどれにも依存しない最適化処理を施させることを特徴とする。

【００１３】請求項４に記載のコンパイラは、バックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）に、中間コードを並列に実行可能な単位に分割させ、ＤＳＰ$_1$乃至ＤＳＰ$_M$に割り当てさせることを特徴とする。

【００１４】請求項５に記載のコンパイラは、バックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）に、レジスタ割付を行わせることを特徴とする。

【００１５】

【作用】請求項１に記載のコンパイル方法においては、ディジタルシグナルプロセッサのプログラムのソースコードを、ディジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のディジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通の中間コードに変換し、その中間コードをディジタルシグナルプロセッサに対応したオブジェクトコードに変換する。従って、ＤＳＰ（ディジタルシグナルプロセッサ）および言語に依存せずに、プログラムをコンパイルすることができる。

【００１６】請求項２に記載のコンパイラにおいては、ＤＳＰ$_1$乃至ＤＳＰ$_M$のプログラムのソースコードを言語Ｌ$_1$乃至Ｌ$_N$に共通の中間コードに変換し、その中間コードを解析して最適化する。そして、最適化した中間コードをＤＳＰ$_1$乃至ＤＳＰ$_M$にそれぞれ対応したオブジェクトコードに変換する。従って、ＤＳＰ（ディジタルシグナルプロセッサ）および言語に依存せずに、プログラムをコンパイルすることができる。

【００１７】請求項３に記載のコンパイラにおいては、共通モジュール部２に、中間コードに対して、ＤＳＰ$_1$乃至ＤＳＰ$_M$のどれにも依存しない最適化処理を施させるので、重複したフェーズによる無駄な処理が防止される。

【００１８】請求項４に記載のコンパイラは、バックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）に、中間コードを並列に実行可能な単位に分割させ、ＤＳＰ$_1$乃至ＤＳＰ$_M$に割り当てさせるので、並列処理をすることができるＤＳＰ（ディジタルシグナルプロセッサ）の機能を有効に利用することができる。

【００１９】請求項５に記載のコンパイラは、バックエンド部３（ＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$）に、レジスタ割付を行わせるので、ＤＳＰ（ディジタルシグナルプロセッサ）のレジスタを効率的に利用することができる。

【００２０】

【実施例】図１は、本発明のコンパイラの一実施例の構成を示すブロック図である。フロントエンド部１は、Ｍ個のディジタルシグナルプロセッサＤＳＰ$_1$乃至ＤＳＰ$_M$（以下、ＤＳＰ$_1$乃至ＤＳＰ$_M$）のいずれかのプログラム記述用言語Ｌ$_1$乃至Ｌ$_N$で記述されたプログラムのソースコードを、言語Ｌ$_1$乃至Ｌ$_N$に共通の中間言語で記述される中間コードにそれぞれ変換する言語Ｌ$_1$用フロントエンド部１$_1$乃至言語Ｌ$_N$用フロントエンド部１$_N$から構成される。各言語Ｌ$_n$用フロントエンド部１$_n$（ｎ＝１，２，・・・，Ｎ）は、図２に示すフェーズ（コンパイルの処理単位）としての、字句解析部１１$_n$、構文解析部１２$_n$、および意味解析部１３$_n$から構成され、言語Ｌ$_n$に依存した処理を行う。

【００２１】即ち、言語Ｌ$_n$用フロントエンド部１$_n$の字句解析部１１$_n$は、言語Ｌ$_n$で記述されたプログラムのソースコードを、論理的に扱うことのできる最小単位の文字列（字句）に分離する。構文解析部１２$_n$は、字句解析部１１$_n$より出力される字句からなる構文を構造解析し、言語Ｌ$_n$の構文構造として許されているか否かを検査（構文チェック）する。意味解析部１３$_n$は、字句からなる構文を意味解析し、構文の意味上の間違いをチェックし、間違いがなければソースコードを、言語Ｌ$_1$乃至Ｌ$_N$のすべてに共通の中間コードに変換する。

【００２２】共通モジュール部２は、図３に示すフェーズとしての依存解析部２１および最適化部２２より構成

され、言語L_1用フロントエンド部1_1乃至言語L_N用フロントエンド部1_Nから出力される中間コードに対して、言語L_1乃至L_NおよびDSP_1乃至DSP_Mに依存しない処理を行う。

【００２３】即ち、共通モジュール部２の依存解析部２１は、フロントエンド部１より出力される中間コードにおける、データの参照または定義の依存関係（データ依存）を解析し、この依存関係（データ依存）を、いわゆるデータフローグラフと呼ばれるデータ構造に変換する。最適化部２２は、必要に応じて依存解析部２１で作成されたデータフローグラフを参照し、フロントエンド部１より出力される中間コードに対して、例えば定数の畳み込みや共通部分式の識別など、言語L_1乃至L_NおよびDSP_1乃至DSP_Mに依存しない最適化処理を施す。

【００２４】バックエンド部３は、共通モジュール部２で最適化された中間コードから、DSP_1乃至DSP_Mに対応したオブジェクトコードをそれぞれ生成するDSP_1用バックエンド部3_1乃至DSP_M用バックエンド部3_Mから構成される。各DSP_m用バックエンド部3_m（m＝１，２，・・・，M）は、図４に示すフェーズとしての、スケジューリング部31_m、コード生成／レジスタ割付部32_m、および最適化部33_mから構成され、DSP_mに依存した処理を行う。

【００２５】即ち、スケジューリング部31_mは、例えばDSP_mのアーキテクチャが並列処理を行うことができるものである場合、共通モジュール部２で最適化された中間コードを、並列に実行可能な単位（タスク）に分割し、DSP_mを構成する、例えばＡＬＵ、乗算器、加算器、またはシフタなどに割り当てる。コード生成／レジスタ割付部32_mは、共通モジュール部２（最適化部２２）より出力される中間コードから、DSP_mに対応したオブジェクトコードを生成するとともに、DSP_mの有するレジスタに、効率的に変数を割り付けるためのレジスタ割付を行う。最適化部33_mは、必要に応じて共通モジュール部２の依存解析部２１で作成されたデータフローグラフを参照し、コード生成／レジスタ割付部32_mで生成されたDSP_mに対応したオブジェクトコードに対して、DSP_mに依存した最適化処理を施す。

【００２６】このように構成されるコンパイラにおいて、言語L_nで記述されたDSP_m用のプログラムがコンパイルされる場合、まずフロントエンド部１の言語L_n用フロントエンド部1_nにそのプログラムが読み込まれる。そして、言語L_n用フロントエンド部1_nの字句解析部11_n（図２）において、言語L_nで記述されたプログラムのソースコードが、例えば手掛かり語（例えば、Ｃ

$$A＝B＋C$$
$$B＝A＋E$$
$$B＝F＋G$$

という３つの式が、式（１），（２），（３）の順番で実行されるように、プログラムが記述されているとする言語やＦＯＲＴＲＡＮでいうところのdo, while, if、およびforなど）、識別子（例えば、プログラムにおける変数など）、定数、並びに演算子（例えば、＋，－，＊、および／など）など、論理的に扱うことのできる最小単位の文字列（字句）に分離される。

【００２７】即ち、字句解析部11_nにおいて、例えば

ＩＦ　（５．ＥＱ．ＭＡＸ）　ＧＯＴＯ　１００

というＦＯＲＴＲＡＮ文は、

ＩＦ，（，５，．ＥＱ．，ＭＡＸ，），ＧＯＴＯ、および１００

という８つの字句に分離される。

【００２８】字句解析部11_nで分離された字句は構文解析部12_nに入力され、構文解析部12_nにおいて、この字句からなる構文が構造解析され、言語L_nの構文構造として許されているか否かが検査（構文チェック）される。即ち、例えば字句Ａ，＋、およびＢが入力された場合、構文解析部12_nにおいて、字句Ａ，＋、およびＢからなる構文Ａ＋Ｂが、式と名付ける構文構造を有すると構造解析され、構文Ａ＋Ｂが言語L_nで記述された式として許されているか否かが検査（構文チェック）される。構文解析部12_nで、任意の構文が、構文チェックで使用不可と判定された場合には、コンパイル（以後の処理）が中止される。

【００２９】構文解析部12_nでの構文チェックをパスした構文は、意味解析部13_nで、意味解析され、構文の意味上の間違いがチェックされる。即ち、意味解析部13_nにおいて、例えば整数型の変数Ａ、演算子＋、および実数型の変数Ｂの３つの字句からなる構文Ａ＋Ｂが、整数型と実数型の加算式であると解析され、整数と実数との加算が間違いか否かがチェックされる。言語L_nの仕様で、整数と実数との加算が許されていなければ、意味解析部13_nで、構文Ａ＋Ｂは使用不可とされ、コンパイル（以後の処理）が中止される。

【００３０】意味解析部13_nで、構文の意味上の間違いがチェックされた構文に間違いがなければ、ソースコードが、言語L_1乃至L_Nのすべてに共通の中間コードに変換され、共通モジュール部２の依存解析部２１（図３）に出力される。

【００３１】依存解析部２１において、意味解析部13_nより出力された中間コードにおける、データの参照または定義の依存関係（データ依存）が解析され、この依存関係（データ依存）が、いわゆるデータフローグラフと呼ばれるデータ構造に変換される。

【００３２】即ち、例えば

（１）
（２）
（３）

と、依存解析部２１では、

・式（１）で定義されたＡが式（２）で参照されている

（フロー依存）。
・式（１）で参照されたＢが式（２）で定義されている（逆依存）。
・式（１）で参照されたＢが式（３）で定義されている（逆依存）。
・式（２）で定義されたＢが式（３）で再定義されている（出力依存）。
のように依存解析がなされ、このデータの参照または定義の依存関係（データ依存）がいわゆるデータフローグラフと呼ばれるデータ構造に変換される。

【００３３】依存解析部２１でデータフローグラフが作成された後、最適化部２２において、必要に応じてこのデータフローグラフが参照され、意味解析部１３$_n$より出力された中間コードに対して、例えば定数の畳み込みや共通部分式の識別など、ＤＳＰ$_1$乃至ＤＳＰ$_M$に依存しない最適化処理が施される。

【００３４】即ち、例えば
Ｉ＝４
Ａ＝Ｉ＊Ｂ
という構文がプログラムの中に言語Ｌ$_n$で記述されていると、最適化部２２において、上記の構文は、
Ａ＝４＊Ｂ
と最適化され、これにより定義（Ｉ＝４）の回数を減らすことができる。

【００３５】また、例えば
Ａ［Ｉ＋１］＝Ｂ［Ｉ＋１］＊Ｃ［Ｉ＋１］
いう構文がプログラムの中に言語Ｌ$_n$で記述されていると、最適化部２２において、上記の構文は、
Ｊ＝Ｉ＋１
Ａ［Ｊ］＝Ｂ［Ｊ］＊Ｃ［Ｊ］
と最適化され、これにより加算（Ｉ＋１）の回数を３回から１回に減らすことができる。

【００３６】共通モジュール部２の最適化部２２で最適化された中間コードは、ＤＳＰ$_m$用バックエンド部３$_m$のスケジューリング部３１$_m$（図４）に出力され、スケジューリング部３１$_m$において、ＤＳＰ$_m$のアーキテクチャが並列処理を行うことができるものである場合には、共通モジュール部２で最適化された中間コードが、並列に実行可能な単位（タスク）に分割され、分割された各タスクが、ＤＳＰ$_m$を構成する、例えばＡＬＵ、乗算器、加算器、またはシフタなどに割り当てられる（スケジューリングされる）。

【００３７】そして、コード生成／レジスタ割付部３２$_m$において、共通モジュール部２の最適化部２２より出力された中間コードから、ＤＳＰ$_m$に対応したオブジェクトコードが生成されるとともに、ＤＳＰ$_m$の有するレジスタに、効率的に変数を割り付けるためのレジスタ割付が行われ、ＤＳＰ$_m$に対応したオブジェクトコードが最適化部３３$_m$に出力される。最適化部３３$_m$において、必要に応じて共通モジュール部２の依存解析部２１で作成されたデータフローグラフが参照され、コード生成／レジスタ割付部３２$_m$より出力されたオブジェクトコードに対して、ＤＳＰ$_m$に依存した最適化処理が施され、言語Ｌ$_n$で記述されたＤＳＰ$_m$用のプログラムのコンパイルが終了する。

【００３８】以下、ＤＳＰ$_m$ で３タップのＦＩＲ型ディジタルフィルタを実現するプログラム

$$y(0)=h(0)*x(0)+h(1)*x(1)+h(2)*x(2)$$
$$y(1)=h(0)*x(1)+h(1)*x(2)+h(2)*x(3)$$
$$y(2)=h(0)*x(2)+h(1)*x(3)+h(2)*x(4)$$

を例にして、ＤＳＰ$_m$用バックエンド部３$_m$のスケジューリング部３１$_m$、コード生成／レジスタ割付部３２$_m$、および最適化部３３$_m$の動作を、さらに説明する。なお、このプログラムにおいて、ｈ（０），ｈ（１）、およびｈ（２）はフィルタの係数、ｘ（０），ｘ（１），・・・はフィルタへの入力信号、ｙ（０），ｙ（１），・・・はフィルタ出力を示す。

【００３９】上記のソースコードを中間コードで表現すると、

```
temp1=h(0)*x(0)                    (a1)
temp2=h(1)*x(1)                    (a2)
temp3=h(2)*x(2)                    (a3)
temp4=temp1+temp2                  (a4)
y(0)=temp3+temp4                   (a5)
temp5=h(0)*x(1)                    (b1)
temp6=h(1)*x(2)                    (b2)
temp7=h(2)*x(3)                    (b3)
temp8=temp5+temp6                  (b4)
y(1)=temp7+temp8                   (b5)
temp9=h(0)*x(2)                    (c1)
temp10=h(1)*x(3)                   (c2)
temp11=h(2)*x(4)                   (c3)
```

temp12＝temp9＋temp10　（c4）
y（2）＝temp11＋temp12　（c5）

となる。

【００４０】ここで、ＤＳＰmが、充分な数のレジスタを有するとすると、式（ａ１）乃至（ａ５）、式（ｂ１）乃至（ｂ５）、および式（ｃ１）乃至（ｃ５）（以下、式（ａ１）乃至（ｃ５）と記載する）のデータ依存は、前述したフロー依存のみとなり、式（ａ１）および式（ａ２）での定義が、式（ａ４）で参照されているというフロー依存を

（ａ１），（ａ２）→（ａ４）

と表すと、式（ａ１）乃至（ａ５）、式（ｂ１）乃至（ｂ５）、および式（ｃ１）乃至（ｃ５）のデータ依存は、

（ａ１），（ａ２）→（ａ４）
（ａ３），（ａ４）→（ａ５）
（ｂ１），（ｂ２）→（ｂ４）
（ｂ３），（ｂ４）→（ｂ５）
（ｃ１），（ｃ２）→（ｃ４）
（ｃ３），（ｃ４）→（ｃ５）

となる。

【００４１】ＤＳＰmが並列に実行可能なアーキテクチャを有さない場合、即ち例えば演算器として２入力１出力の演算器を１つだけＤＳＰmが有する場合、スケジューリング部３１mにおいて、式（ａ１）乃至（ｃ５）で示される演算が、ＤＳＰmが内蔵する１つだけの演算器に、図５に示すように割り当てられる（スケジューリングされる）。

【００４２】即ち、ＤＳＰmが内蔵する演算器が１クロックで動作するとすると、スケジューリング部３１mでは、式（ａ１）乃至（ｃ５）で示される演算が、それぞれ１乃至１５クロック目に、演算器で行われるように割り当てられる。

【００４３】また、ＤＳＰmが並列に実行可能なアーキテクチャを有する場合、即ち例えば演算器として、９個の、２入力１出力の乗算器X_1乃至X_9、および６個の、２入力１出力の加算器Y_1乃至Y_6をＤＳＰmが有する場合、スケジューリング部３１mにおいて、式（ａ１）乃至（ｃ５）で示される演算が、ＤＳＰmが内蔵する乗算器X_1乃至X_9および加算器Y_1乃至Y_6に、図６に示すように割り当てられる（スケジューリングされる）。

【００４４】即ち、乗算器X_1乃至X_9および加算器Y_1乃至Y_6が１クロックで動作するとすると、スケジューリング部３１mにおいて、１クロック目に、式（ａ１）乃至（ａ３）、式（ｂ１）乃至（ｂ３）、または式（ｃ１）乃至（ｃ３）で示される演算（乗算）が、乗算器X_1乃至X_9でそれぞれ行われるように割り当てられ、２クロック目に、式（ａ４），（ｂ４）、または（ｃ４）で示される演算（加算）が、加算器Y_1，Y_3、またはY_5でそれぞれ行われるように割り当てられるとともに、３クロック目に、式（ａ５），（ｂ５）、または（ｃ５）で示される演算（加算）が、加算器Y_2，Y_4、またはY_6でそれぞれ行われるように割り当てられる。

【００４５】さらに、ＤＳＰmが、例えば演算器として、３個の、２入力１出力の乗算器X_1乃至X_3、並びに２個の、２入力１出力の加算器Y_1およびY_2を有する場合、スケジューリング部３１mにおいて、式（ａ１）乃至（ｃ５）で示される演算が、ＤＳＰmが内蔵する乗算器X_1乃至X_3並びに加算器Y_1およびY_2に、図７に示すように割り当てられる（スケジューリングされる）。

【００４６】即ち、乗算器X_1乃至X_3並びに加算器Y_1およびY_2が１クロックで動作するとすると、スケジューリング部３１mにおいて、１クロック目に、式（ａ１）乃至（ａ３）で示される演算が、乗算器X_1乃至X_3でそれぞれ行われるように割り当てられ、２クロック目に、式（ｂ１）乃至（ｂ３）、または式（ａ４）で示される演算が、乗算器X_1乃至X_3、または加算器Y_1でそれぞれ行われるように割り当てられるとともに、３クロック目に、式（ｃ１）乃至（ｃ３）、式（ｂ４）、または式（ａ５）で示される演算が、乗算器X_1乃至X_3、加算器Y_1またはY_2でそれぞれ行われるように割り当てられる。さらに、４クロック目には、式（ｃ４）または式（ｂ５）で示される演算が、加算器Y_1またはY_2でそれぞれ行われるように割り当てられ、５クロック目に、式（ｃ５）で示される演算が、加算器Y_2で行われるように割り当てられる。

【００４７】また、ＤＳＰmが、例えば演算器として、２個の、２入力１出力の乗算器X_1およびX_2、並びに１個の、２入力１出力の加算器Y_1を有する場合、スケジューリング部３１mにおいて、式（ａ１）乃至（ｃ５）で示される演算が、ＤＳＰmが内蔵する乗算器X_1およびX_2並びに加算器Y_1に、図８に示すように割り当てられる（スケジューリングされる）。

【００４８】即ち、乗算器X_1およびX_2並びに加算器Y_1が１クロックで動作するとすると、スケジューリング部３１mにおいて、乗算器X_1で、式（ａ１），（ａ３），（ｂ２），（ｃ１）、または（ｃ３）で示される演算が、１乃至５クロック目にそれぞれ行われるように割り当てられ、乗算器X_2で、式（ａ２），（ｂ１），（ｂ３）、または（ｃ２）で示される演算が、１乃至４クロック目にそれぞれ行われるように割り当てられるとともに、加算器Y_1で、式（ａ４），（ａ５），（ｂ４），（ｂ５），（ｃ４）、または（ｃ５）で示される演算が、２乃至７クロック目にそれぞれ行われるように割り当てられる。

【００４９】スケジューリング部３１mでのスケジューリングが終了すると、コード生成／レジスタ割付部３２mにおいて、ＤＳＰmの有するレジスタに、効率的に変数

を割り付けるためのレジスタ割付が行われながら、式（ａ１）乃至式（ｃ５）に示す中間コードから、ＤＳＰ$_m$に対応したオブジェクトコードが生成される。

【００５０】ＤＳＰ$_m$が３つのレジスタｒｅｇ１乃至ｒｅｇ３を有する場合、コード生成／レジスタ割付部３２

```
MUL  h (0), x (0), reg1
MUL  h (1), x (1), reg2
MUL  h (2), x (2), reg3
ST   reg3, mem1
ADD  reg1, reg2, reg3
LD   mem1, reg1
ADD  reg1, reg3, y (0)
MUL  h (0), x (1), reg1
MUL  h (1), x (2), reg2
MUL  h (2), x (3), reg3
ST   reg3, mem2
ADD  reg1, reg2, reg3
LD   mem2, reg1
ADD  reg1, reg3, y (1)
MUL  h (0), x (2), reg1
MUL  h (1), x (3), reg2
MUL  h (2), x (4), reg3
ST   reg3, mem3
ADD  reg1, reg2, reg3
LD   mem3, reg1
ADD  reg1, reg3, y (2)
```

のように生成される。

【００５１】オブジェクトコード（Ａ１）乃至（Ａ３）は、

ｒｅｇ１＝ｈ（０）＊ｘ（０），

ｒｅｇ２＝ｈ（１）＊ｘ（１）、または

ｒｅｇ３＝ｈ（２）＊ｘ（２）

をそれぞれ示し、式（ａ１）乃至（ａ３）に対応する。ＤＳＰ$_m$には、３つのレジスタｒｅｇ１乃至ｒｅｇ３しかないので、式（ａ４）を計算するために、オブジェクトコード（Ａ４）で、レジスタｒｅｇ３の内容（式（ａ３）の変数ｔｅｍｐ３の内容）がスタックｍｅｍ１に退避され（ストアされ）、オブジェクトコード（Ａ５）で、式（ａ４）に対応する

ｒｅｇ３＝ｒｅｇ１＋ｒｅｇ２

が計算される。

【００５２】そして、オブジェクトコード（Ａ６）で、スタックｍｅｍ１にストアされた式（ａ３）の変数ｔｅｍｐ３の内容（ｈ（２）＊ｘ（２））が、レジスタｒｅｇ１にロードされ、オブジェクトコード（Ａ７）で、式（ａ５）に対応する

ｙ（０）＝ｒｅｇ１＋ｒｅｇ３

が計算される。

【００５３】オブジェクトコード（Ｂ１）乃至（Ｂ７）、または（Ｃ１）乃至（Ｃ７）でも、上記したオブジェクトコード（Ａ１）乃至（Ａ７）における場合と同様にして、式（ｂ１）乃至（ｂ５）、または（ｃ１）乃至（ｃ５）にそれぞれ対応する処理がなされる。

【００５４】なお、ＤＳＰ$_m$が、例えば１クロックで動作する演算器として、３個の、２入力１出力の乗算器Ｘ$_1$乃至Ｘ$_3$、並びに２個の、２入力１出力の加算器Ｙ$_1$およびＹ$_2$を有し、データのロードおよびストアを１クロックで行うとすると、上記のオブジェクトコード（Ａ１）乃至（Ａ７），（Ｂ１）乃至（Ｂ７）、および（Ｃ１）乃至（Ｃ７）は、図９に示すように実行される。

【００５５】即ち、１クロック目に、オブジェクトコード（Ａ１）乃至（Ａ３）に対応する式（ａ１）乃至（ａ３）で示される演算が、乗算器Ｘ$_1$乃至Ｘ$_3$でそれぞれ行われ、２クロック目に、オブジェクトコード（Ｂ１）乃至（Ｂ３）に対応する式（ｂ１）乃至（ｂ３）で示される演算が、乗算器Ｘ$_1$乃至Ｘ$_3$でそれぞれ行われるとともに、オブジェクトコード（Ａ４）に対応するレジスタｒｅｇ３のスタックｍｅｍ１へのストアが行われる。

【００５６】３クロック目に、オブジェクトコード（Ｃ１）乃至（Ｃ３）または（Ａ５）に対応する式（ｃ１）乃至（ｃ３）、または式（ａ４）で示される演算が、乗算器Ｘ$_1$乃至Ｘ$_3$、加算器Ｙ$_1$でそれぞれ行われるとともに、オブジェクトコード（Ｂ４）に対応するレジスタｒｅｇ３のスタックｍｅｍ２へのストアが行われ、４クロ

ック目に、オブジェクトコード（Ｂ５）に対応する式（ｂ４）で示される演算が、加算器Y_1で行われるとともに、オブジェクトコード（Ｃ４）に対応するレジスタｒｅｇ３のスタックｍｅｍ３へのストアと、オブジェクトコード（Ａ６）に対応するスタックｍｅｍ１からレジスタｒｅｇ１へのロードが行われる。

【００５７】５クロック目には、オブジェクトコード（Ｃ５）または（Ａ７）に対応する式（ｃ４）または（ａ５）で示される演算が、加算器Y_1またはY_2でそれぞれ行われるとともに、オブジェクトコード（Ｂ６）に対応するスタックｍｅｍ２からレジスタｒｅｇ１へのロードが行われ、６クロック目に、オブジェクトコード

ＬＤ　ｍｅｍ，ｒｅｇ
ＳＴ　ｒｅｇ，ｍｅｍ

のような、メモリｍｅｍからレジスタｒｅｇにデータをロードし（Ｄ１）、すぐにレジスタｒｅｇのデータをメモリｍｅｍにストアする（Ｄ２）ことを示すオブジェクトコードにおいては、同じ内容（データ）を、メモリｍ

ＡＤＤ　０，ｒｅｇ，ｒｅｇ
ＭＵＬ　１，ｒｅｇ，ｒｅｇ

のような、レジスタｒｅｇに０を加算して、レジスタｒｅｇに記憶させたり（Ｅ１）、レジスタｒｅｇに１を乗じて、レジスタｒｅｇに記憶させたりする（Ｅ２）オブジェクトコードにおいては、０の加算や１の乗算は結果が変わらないので、やはり無駄であるから、最適化部３３$_m$において、オブジェクトコード（Ｅ１）および（Ｅ１）とも削除される（代数的簡約化）。

【００６２】以上のように、このコンパイラでは、プログラムを記述した言語L_1乃至L_Nに対応した言語L_1用フロントエンド部１$_1$乃至言語L_Nフロントエンド部１$_N$で前処理を行い、ＤＳＰ$_1$乃至ＤＳＰ$_M$に対応したオブジェクトコードを出力するＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$で後処理を行うとともに、言語L_1乃至L_NおよびＤＳＰ$_1$乃至ＤＳＰ$_M$に依存しない処理（フェーズ）を共通モジュール部２で行うようにしたので、言語L_1乃至L_Nのうちのどの言語で記述されたプログラムでも、また、ＤＳＰ$_1$乃至ＤＳＰ$_M$のうちのどのＤＳＰに対応するプログラムでもコンパイルすることができる。

【００６３】

【発明の効果】請求項１に記載のコンパイル方法によれば、ディジタルシグナルプロセッサ（ＤＳＰ）のプログラムのソースコードを、ＤＳＰのプログラムを記述するための複数の言語、または複数のＤＳＰにそれぞれ対応したオブジェクトコードに共通の中間コードに変換し、その中間コードをＤＳＰに対応したオブジェクトコードに変換する。従って、ＤＳＰおよび言語に依存せずに、プログラムをコンパイルすることができる。

【００６４】請求項２に記載のコンパイラによれば、ディジタルシグナルプロセッサ（ＤＳＰ）のプログラムの（Ｂ７）に対応する式（ｂ５）で示される演算が、加算器Y_2で行われるとともに、オブジェクトコード（Ｃ６）に対応するスタックｍｅｍ３からレジスタｒｅｇ１へのロードが行われる。

【００５８】そして、７クロック目に、オブジェクトコード（Ｃ７）に対応する式（ｃ５）で示される演算が、加算器Y_2で行われ、処理を終了する。

【００５９】次に、上記のような、ＤＳＰ$_m$に対応したオブジェクトコードが、最適化部３３$_m$に供給されると、最適化部３３$_m$において、このオブジェクトコードに対して、ＤＳＰ$_m$に依存した最適化処理が施される。

【００６０】即ち、例えば

（Ｄ１）
（Ｄ２）

ｅｍにストアすることは無駄であるから、最適化部３３$_m$において、オブジェクトコード（Ｄ２）が削除される（冗長なロード／ストア命令の削除）。

【００６１】また、例えば

（Ｅ１）
（Ｅ２）

ソースコードを言語に共通の中間コードに変換し、その中間コードを解析して最適化する。そして、最適化した中間コードを複数のＤＳＰにそれぞれ対応したオブジェクトコードに変換する。従って、ＤＳＰおよび言語に依存せずに、プログラムをコンパイルすることができる。

【００６５】請求項３に記載のコンパイラによれば、共通処理手段に、中間コードに対して、複数のＤＳＰのどれにも依存しない最適化処理を施させるので、重複したフェーズによる無駄な処理が防止される。

【００６６】請求項４に記載のコンパイラは、後処理手段に、中間コードを並列に実行可能な単位に分割させ、ＤＳＰに割り当てさせるので、並列処理をすることができるＤＳＰの機能を有効に利用することができる。

【００６７】請求項５に記載のコンパイラは、後処理手段に、レジスタ割付を行わせるので、ＤＳＰのレジスタを効率的に利用することができる。

【図面の簡単な説明】

【図１】本発明のコンパイラの一実施例の構成を示すブロック図である。

【図２】図１の実施例の言語L_1用フロントエンド部１$_1$乃至言語L_Nフロントエンド部１$_N$のより詳細を示す図である。

【図３】図１の実施例の共通モジュール部２のより詳細を示す図である。

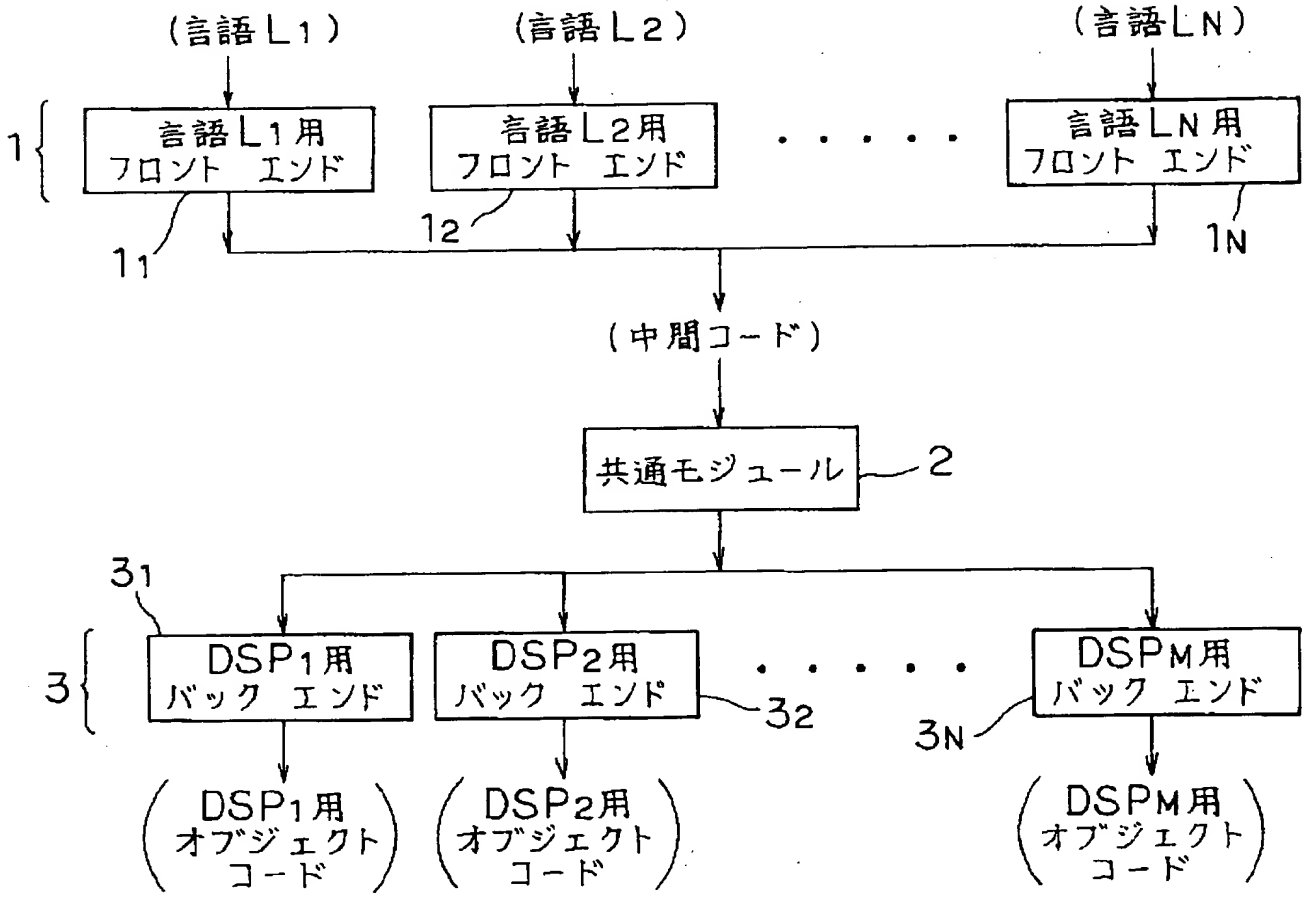
【図４】図１の実施例のＤＳＰ$_1$用バックエンド部３$_1$乃至ＤＳＰ$_M$用バックエンド部３$_M$のより詳細を示す図である。

【図５】図４のスケジューリング部で行われるスケジューリングを説明するための図である。

【図６】図４のスケジューリング部で行われるスケジュ

ーリングを説明するための図である。
【図７】図４のスケジューリング部で行われるスケジューリングを説明するための図である。
【図８】図４のスケジューリング部で行われるスケジューリングを説明するための図である。
【図９】図４のコード生成／レジスタ割付部３２ₘで生成されたＤＳＰₘに対応したオブジェクトコードが実行される様子を説明するための図である。
【符号の説明】
１　フロントエンド部
１₁乃至１ₙ　言語Ｌ₁用フロントエンド部乃至言語Ｌ₮フロントエンドＮ部
２　共通モジュール部
３　バックエンド部
３₁乃至３ₘ　ＤＳＰ₁用バックエンド部乃至ＤＳＰₘ用バックエンド部
１１ｎ　字句解析部
１２ｎ　構文解析部
１３ｎ　意味解析部
２１　依存解析部
２２　最適化部
３１ｍ　スケジューリング部
３２ｍ　コード生成／レジスタ割付部
３３ｍ　最適化部

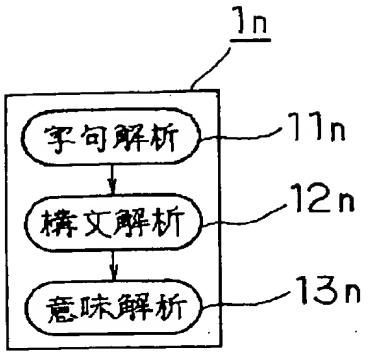
【図１】



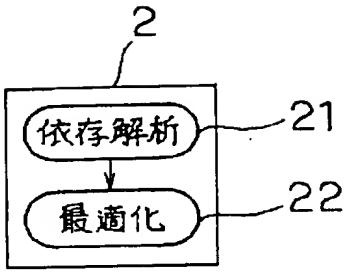
【図５】

クロック	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
演算器	(a1)	(a2)	(a3)	(a4)	(a5)	(b1)	(b2)	(b3)	(b4)	(b5)	(c1)	(c2)	(c3)	(c4)	(c5)

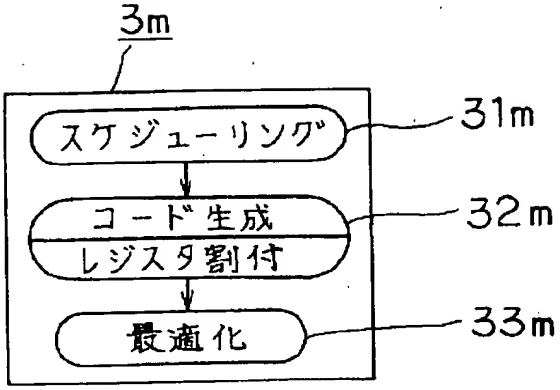
【図2】



【図3】



【図4】



【図6】

クロック	1	2	3
乗算器 X_1	(a1)		
乗算器 X_2	(a2)		
乗算器 X_3	(a3)		
加算器 Y_1		(a4)	
加算器 Y_2			(a5)
乗算器 X_4	(b1)		
乗算器 X_5	(b2)		
乗算器 X_6	(b3)		
加算器 Y_3		(b4)	
加算器 Y_4			(b5)
乗算器 X_7	(c1)		
乗算器 X_8	(c2)		
乗算器 X_9	(c3)		
加算器 Y_5		(c4)	
加算器 Y_6			(c5)

【図7】

クロック	1	2	3	4	5
乗算器 X_1	(a1)	(b1)	(c1)		
乗算器 X_2	(a2)	(b2)	(c2)		
乗算器 X_3	(a3)	(b3)	(c3)		
加算器 Y_1		(a4)	(b4)	(c4)	
加算器 Y_2			(a5)	(b5)	(c5)

【図8】

クロック	1	2	3	4	5	6	7
乗算器 X_1	(a1)	(a3)	(b2)	(c1)	(c3)		
乗算器 X_2	(a2)	(b1)	(b3)	(c2)			
加算器 Y_1		(a4)	(a5)	(b4)	(b5)	(c4)	(c5)

【図9】

クロック	1	2	3	4	5	6	7
乗算器 X_1	(A1)	(B1)	(C1)				
乗算器 X_2	(A2)	(B2)	(C2)				
乗算器 X_3	(A3)	(B3)	(C3)				
加算器 Y_1			(A5)	(B5)	(C5)		
加算器 Y_2					(A7)	(B7)	(C7)
ストア		(A4)	(B4)	(C4)			
ロード				(A6)	(B6)	(C6)	